iWebShop 插件开发手册

V4.4

2016-03-20

目录

概述

此文档仅适合对 iWebShop 整体架构和开发模式比较了解，并且有一定 php 技术经验的开发人员，此文档专业性比较强，不适合非技术类人员阅读。

iWebShop 从 V4.4 版本开始全面支持插件机制，插件是什么东西？总体来说，插件是一种可以热插拔的（动态安装和卸载），可以实现一定功能性并且对目前现有运行系统不会产生任何影响的一种松散耦合的设计模式，它内容丰富，而且易扩展，可以有更多的开发者参与进来，让产品自身的功能更加丰富多彩，它也可以通过动态的安装组合，实现不同的产品架构。

拦截器（钩子）

说到插件就不得不提到钩子，插件之所以能够动态的加载到系统运行的各个环节中，主要是依赖于钩子，什么是钩子呢？钩子就是 iWebShop 系统在运行中预留的一些事件函数，我们把插件注册（挂接）到 iWebShop 的某个事件上面，当 iWebShop 运行到这个事件的时候就会自动触发并且调用插件，只有明白了这个原理，我们才能继续下面的开发，关于更多钩子机制，可以百度学习一下。

iWebShop 为开发者预留的大量的钩子，同时也支持开发者自己定义钩子事件，下面介绍一下系统都预留了哪些钩子可以让开发者使用。这些钩子的调用和触发都在插件的核心管理类中，下面会重点介绍！

插件核心类 plugin.php

classes/plugin.php 是插件的核心类，所有插件的注册，调用，配置等等都必须要依靠它！

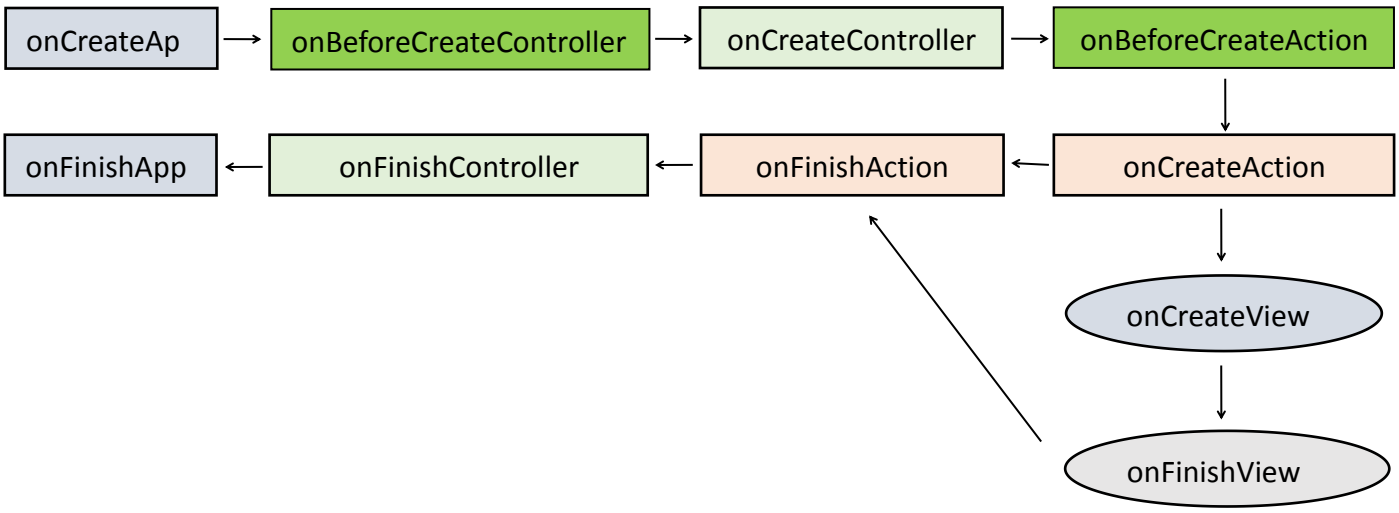
插件内置钩子事件

在核心插件类里面有系统内置的所有钩子，如图：

```
/**
 * @brief 系统内置的所有事件触发
 */
public static function onCreateApp(){plugin::init();plugin::trigger("onCreateApp");}
public static function onFinishApp(){plugin::trigger("onFinishApp");}

public static function onBeforeCreateController($ctrlId){plugin::trigger("onBeforeCreateController");plugin::trigger
public static function onCreateController($ctrlObj){plugin::trigger("onCreateController");plugin::trigger("onCreateC
public static function onFinishController($ctrlObj){plugin::trigger("onFinishController");plugin::trigger("onFinishC

public static function onBeforeCreateAction($ctrlObj,$actionId){plugin::trigger("onBeforeCreateAction");plugin::trig
public static function onCreateAction($ctrlObj,$actionObj){plugin::trigger("onCreateAction");plugin::trigger("onCrea
public static function onFinishAction($ctrlObj,$actionObj){plugin::trigger("onFinishAction");plugin::trigger("onFini

public static function onCreateView($ctrlObj,$actionObj){plugin::trigger("onCreateView");plugin::trigger("onCreateVi
public static function onFinishView($ctrlObj,$actionObj){plugin::trigger("onFinishView");plugin::trigger("onFinishVi

public static function onPhpShutDown(){plugin::trigger("onPhpShutDown");}
```

整个系统会在各自的地方调用预留的钩子，这些钩子就是让插件工作起来的源动力！

iWebShop 在运行过程中，会自动触发 plugin 核心类的注册事件，以达到调用各个插件的目的。



以上流程图就是 iWebShop 整个运行所触发的事件，我们的插件就要注册（挂接）到这些事件上，插件里面的代码可以在绑定的事件下运行，而且还可以给任意的控制器（controller）里面扩展动作（action）。让插件里面的代码也可以和系统原有的控制器动作一样运行。

事件触发是广泛的，比如 onCreateController 这是事件会监听所有控制器的创建，但是有的时候，我们的插件仅仅是针对某个控制器的，比如前台访问统计，或者是后台的订单自动取消等，这些功能仅需要监听（注册）site.php 和 order.php 控制器，于是我们可以把事件名称做一下变形，比如：onCreateController@site　onCreateController@order 这样就可以为某个控制器绑定事件了！

如果我们要在 simple 控制器下面扩展一个 AAA 的动作方法，那么我们就要在 onBeforeCreateAction@simple@AAA 绑定这个事件，这个事件的解释：在创建 simple 控制器下的 AAA 动作之前调用，就是在程序要调用 simple 下的 AAA 的时候，我们动态的给当前控制器（simple）增加方法。

```
plugin::reg("onBeforeCreateAction@simple@AAA",function(){
    self::controller()->AAA = function(){echo "hello AAA";}
});
```

同理动作（action）也是如此，那么我们只需要继续追加@动作 ID　比如，我们要在首页（site/index）动作中做处理，那么监听的事件名字：onCreateAction@site@index 按照层级关系追加就可以了，控制器（controller）包含动作（action）。

下面我们介绍最常用的事件：

❖ 【onBeforeCreateAction】重点!!

动作（action）创建之前，此时控制器已经创建了，动作（action）还没有创建，利用此方法可以动态扩展控制器下的动作，让插件的代码可以整合到控制器里面被程序调用。

事件名称后面直接追加@控制器 ID@动作 ID 可以在特定的动作调用之前进行扩展，比如有个插件需要在 site 控制器下，增加个 PPP 动作方法，用来输出“iWebShop4.4”这个字符串，那么我们就需要在插件的 reg 接口里面编写：

```
plugin::reg("onBeforeCreateAction@site@PPP",function(){
    self::controller()->PPP = function(){echo "iWebShop4.4";}
});
```

当然也可以绑定到一个插件类方法上 self::controller()-PPP=$this->aaa();

通过以上方式，把原本不存在的 PPP 方法，动态添加到了 site 控制器里面，而且对其他程序无任何影响！只有在程序要调用 site/PPP 的时候才会运行！是不是很酷

❖ 【onCreateController】

控制器（controller）创建完成，可以得到当前正在运行的控制器，此时动作（action）还没有创

建。

在事件名称后面追加@控制器 ID 可以对控制器做监听。

- 【onCreateAction】
  动作（action）创建完成，此时已经执行完动作（action）
  在事件名称后面追加@控制器 ID 或者 @控制器 ID@动作 ID 可以对控制器或者动作做监听。

- 【onCreateView】
  视图（view）创建完成
  在事件名称后面追加@控制器 ID 或者 @控制器 ID@动作 ID 可以对单独控制器或者动作做监听。

- 【onFinshView】
  视图（view）调用结束
  在事件名称后面追加@控制器 ID 或者 @控制器 ID@动作 ID 可以对单独控制器或者动作做监听。

- 【onFinishAction】
  动作（action）调用结束
  在事件名称后面追加@控制器 ID 或者 @控制器 ID@动作 ID 可以对单独控制器或者动作做监听。

- 【onFinishController】
  控制器完成结束
  在事件名称后面追加@控制器 ID 可以对单独控制器或者动作做监听。

插件注册接口 reg

public static function reg($event,$classObj,$method = "")
$event ： 事件名称;
$classObj：类对象或者匿名函数
$method：方法名称，当$classObj 为类对象时此参数有意义，否则无用
核心插件类中有一个重要的接口 通过此静态方法，把事件与插件进行注册绑定（挂接），简单说就是把某个$event 事件交由一个函数或者对象方法去处理。

```
11  class orderAutoUpdate extends pluginBase
12  {
13      //注册事件
14      public function reg()
15      {
16          plugin::reg("onCreateAction@order@order_list",$this,"orderUpdate");
17          plugin::reg("onCreateAction@ucenter@order",$this,"orderUpdate");
18      }
19
```

当系统运行 order/order_list 的时候，运行自定义插件里面的 orderUpdate 方法。

事件触发接口 trigger

public static function trigger($event,$data = null)
$event : 事件名称
$data : 传送参数
当系统运行到每个事件的时候，都要运行此接口，用来通知各个插件，由于插件父类（pluginBase）已经预留了以上的一些系统级事件（比如：onCreateController），并且会自动调用，所以开发者注册

此类系统事件不需要手动调用 trigger 函数，但是如果是非系统事件就必须要在代码中手动触发，比如 classes/menu.php 中的创建后台菜单事件，就必须要调用 plugin::trigger 接口，如图：

```php
    /**
     * @brief 根据权限初始化菜单
     * @param int $roleId 角色ID
     * @return array 菜单数组
     */
    public static function init($roleId)
    {
        //菜单创建事件触发
        plugin::trigger("onSystemMenuCreate");

        //根据角色分配权限
        if($roleId == 0)
        {
            $adminRights = 'administrator';
        }
        else
        {
            $roleObj = new IModel('admin_role');
            $where   = 'id = '.$roleId.' and is_del = 0';
            $roleRow = $roleObj->getObj($where);
            $adminRights = isset($roleRow['rights']) ? $roleRow['rights'] : '';
        }

        //1,超管返回全部菜单
        if($adminRights == "administrator")
        {
            return self::$menu;
        }
```

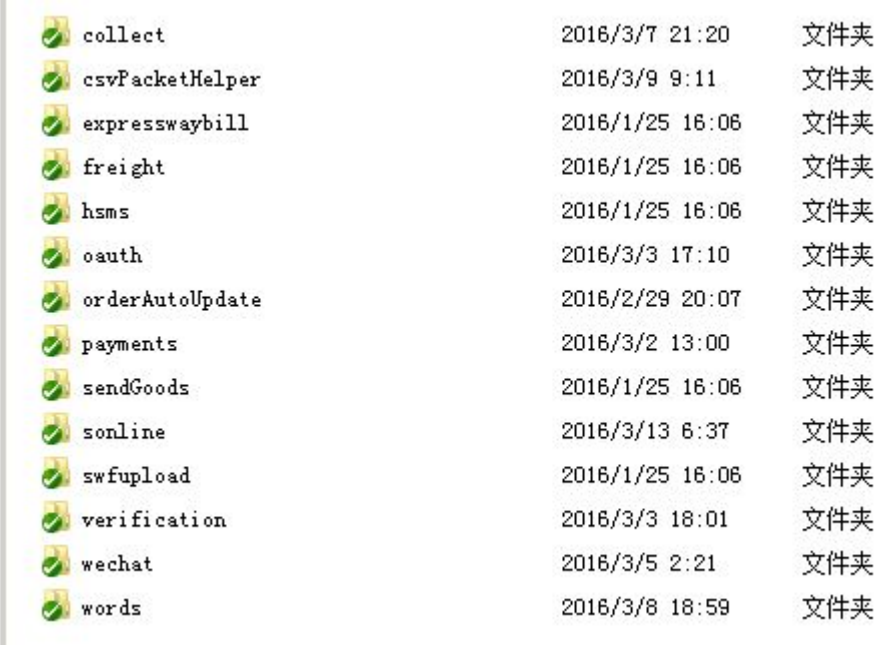
插件基类公开接口

核心文件中有个重要的插件基类 class pluginBase 用户开发的所有插件都必须要继承（extends）这个类，这个插件基类包含了很多工具函数，都是开发者要用到的，所以熟悉和了解此类库对开发插件具有重大意义，这些接口随着日后的更新也会越来越完善。

接口名称	返回参数	接口描述
function config()	数组 array	获取插件的自定义配置参数 configName 存进数据库的数据
public function redirect($view,$data = array()) $view：插件视图名字 $data：渲染的数据，会被控制器自动调用 setRenderData 把数组差分拆变量	显示完整视图	把插件自定义的视图渲染到当前正在运行的控制器里面，整个网页是带着当前控制器布局（layout）一起呈现
public function view($view,$data = array()) $view:插件视图名字 $data:渲染的数据，会被控制器自动调用 setRenderData 把数组差分拆变量	仅显示模板部分	把插件自定义的视图渲染到当前正在运行的控制器里面，不带当前控制器的布局（layout），仅显示插件的单独模板
public function path()	路径字符串 string	获取当前插件所在系统中的物理路径
public function webPath()	路径字符串 string	获取当前插件所在系统中的 URL 网页路径（WEB 地址）
public static controller()	控制器对象	获取当前正在执行的控制器实例
public static action()	动作对象	获取当前正在执行的动作实例

插件存放目录

iWebShop 的所有插件都存放到 plugins 目录下，其下的每个子目录代表每个插件，并且每个插件目录下面还要新建一个与插件目录名称一样的插件入口文件，比如插件名字叫“collect”，那么我们要在 plugins 目录下新建一个 collect 目录，并且在这个 collect 目录下还要新建一个 collect.php 类文件



collect	2016/3/7 21:20	文件夹
csvPacketHelper	2016/3/9 9:11	文件夹
expresswaybill	2016/1/25 16:06	文件夹
freight	2016/1/25 16:06	文件夹
hsms	2016/1/25 16:06	文件夹
oauth	2016/3/3 17:10	文件夹
orderAutoUpdate	2016/2/29 20:07	文件夹
payments	2016/3/2 13:00	文件夹
sendGoods	2016/1/25 16:06	文件夹
sonline	2016/3/13 6:37	文件夹
swfupload	2016/1/25 16:06	文件夹
verification	2016/3/3 18:01	文件夹
wechat	2016/3/5 2:21	文件夹
words	2016/3/8 18:59	文件夹

名称	修改日期	类型	大小
collect.php	2016/3/10 17:32	UltraEdit Docu...	11 KB
sellerCollectImport.html	2016/3/8 15:13	HTML 文档	4 KB
sellerGoodsDetail.html	2016/3/7 21:31	HTML 文档	1 KB
systemCollectImport.html	2016/3/7 21:14	HTML 文档	3 KB
systemGoodsDetail.html	2016/3/7 18:28	HTML 文档	1 KB

我们把每个插件所需要的所有东西都放到自己的目录里面。

开发插件类文件

在插件目录下必须要有一个与其同名的类文件，我们称之为“插件入口文件”，所有的规范都要在这个入口文件中体现出来，插件开发最重要的部分马上就要来了~插件入口必须满足以下开发规范：

1. 必须继承（extends）pluginBase 插件基类；
2. 必须实现如下静态方法（static），用于后台插件列表提取必要信息：

静态方法名称	方法用途
public static function name(){return ”插件名字”;}	【必填】插件中文名称 【返回值】字符串（string）
public static function description(){return “插件简介”;}	【必填】插件的功能简介 【返回值】字符串（string）
public static function explain(){return “插件用法”;}	【选填】插件需要使用者具体用法，如果是静默无感知的插件，就可以省略

	【返回值】字符串（string）
public static function install(){安装插件代码...return true;}	【选填】如果插件需要数据库支持，可以在这里新建数据表 【返回值】布尔值（boolean）
public static function uninstall(){卸载插件代码...return true;}	【选填】把安装时创建的数据表删除，去掉冗余的数据 【返回值】布尔值（boolean）
public static function configName(){return array(插件参数配置);}	【选填】开发者自定义插件的配置参数，可以在系统后台插件修改中进行选择，并且设置的参数也可以在插件运行过程中进行读取应用 【返回值】数组（array）

这里把上面的静态接口做简单说明：

name，description，explain 这 3 个静态方法都比较简单，就是返回字符串就可以了，信息显示如图

插件 > 插件管理 > 插件列表

名称	描述	状态
商品云采集	支持采集JD（京东）列表和单品页面信息，支持商品详情采集	已关闭【已安装】
商品CSV数据导入	由淘宝助手或者用手工具箱产生的CSV数据包直接导入到iWebShop系统中	已关闭【已安装】
订单自动完成和取消	订单自动完成和取消。1，已经发货的订单会在X分钟后自动完成；2，未付款的订单会在X分钟后自动取消	已关闭【未安装】
QQ客服插件	网站和商家可以分别设置多个QQ客服，在商家主页，商品详情页面都有QQ咨询按钮	使用中【已安装】
微信插件	微信免登录免注册，微信支付，公众账号菜单定制，js-sdk对接	使用中【已安装】
SCWS商品分词插件	1,商品添加修改时对名称进行分词; 2,商品列表查询分词	使用中【已安装】

install 方法里面是安装插件时候要往 iWebShop 系统中做的更新操作代码，比如要创建一些文件，创建数据表，更新一些数据等，比如 sonline（QQ 客服插件），就需要创建数据表来存储商家和管理员的客服 QQ 号码等数据，那么 sonline 插件的 install 里面就是如图：

```php
class sonline extends pluginBase
{
    public static function name()
    {
        return "QQ客服插件";
    }

    public static function description()
    {
        return "网站和商家可以分别设置多个QQ客服，在商家主页，商品详情页面都有QQ咨询按钮";
    }

    public static function install()
    {
        $onlineDB = new IModel('online_service');
        $data = array(
            "comment" => self::name(),
            "column"  => array(
                "id" => array("type" => "int(11) unsigned",'auto_increment' => 1),
                "qq" => array("type" => "text","comment" => "客服QQ名称和号码json数据格式"),
                "seller_id" => array("type" => "int(11) unsigned","default" => "0","comment" => "商家ID，如果为0表示平台自营客服
            ),
            "index" => array("primary" => "id","key" => "seller_id"),
        );
        $onlineDB->setData($data);
        return $onlineDB->createTable();
    }

    public static function uninstall()
    {
        $onlineDB = new IModel('online_service');
        return $onlineDB->dropTable();
    }
```

这里的install方法里面直接调用IModel类库创建数据表，关于IModel方面的操作，可以阅读《iWebShop内核技术手册》这里不再赘述。方法最后返回的 boolean 值，如果返回 true 状态，则系统会成功安装，否则表示有错误。

uninstall 方法同理，里面编写卸载插件所需要处理的代码，方法最后返回的 boolean 值，如果返回 true 状态，则系统会成功安装，否则表示有错误。

configName 插件参数配置，比如后台管理员要修改一些配置，然后前台的插件会显示不同的状态或者功能，数组的具体格式分为：

```
array("字段名" => array(
    "name"    => "文字显示",
    "type"    => "数据类型【text,radio,checkbox,select】",
    "pattern" => "数据校验【int,float,date,datetime,require,正则表达式】",
    "value"   => "1,数组：枚举数据【radio,checkbox,select】的预设值,array(名字=>数据);
                  2,字符串：【text】默认数据",
))
```

还是用 soline 客服插件做例子：

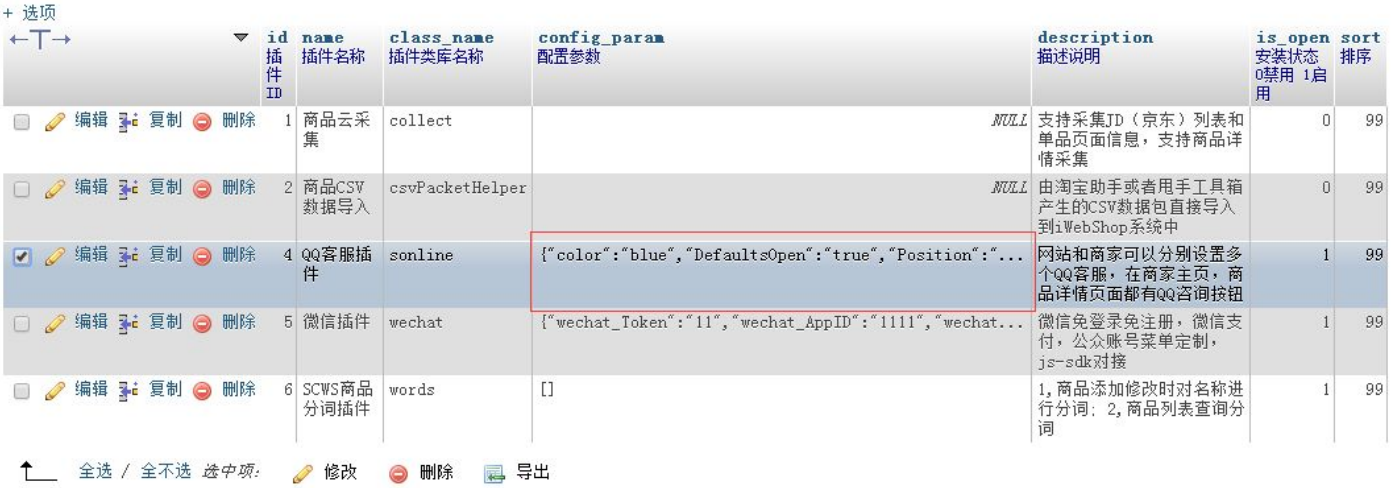
```
public static function configName()
{
    return array(
        "color" => array("name" => "颜色风格","type" => "select","value" => array("蓝色" => "blue","红色" => "red","
        "DefaultsOpen" => array("name" => "显示方式","type" => "select","value" => array("展开" => "true","隐藏" =>
        "Position" => array("name" => "显示位置","type" => "select","value" => array("左侧" => "left","右侧" => "rig
    );
}
```

我们看到了配置数组里面有 3 个元素，color（颜色风格），DefaultsOpen（显示方式），Position（显示位置），然后设置了 UI 的选择类型，和默认的选择值，那么进入后台插件列表

然后点击“QQ 客服插件”编辑按钮就看到了配置

点击保存后，就会存到 iwebshop 的 plugin 表里面（这里的 plugin 是系统自带的表，专门用于记录插

件信息）



+ 选项

	id 插件ID	name 插件名称	class_name 插件类库名称	config_param 配置参数	description 描述说明	is_open 安装状态 0禁用 1启用	sort 排序
编辑 复制 删除	1	商品云采集	collect	NULL	支持采集JD（京东）列表和单品页面信息，支持商品详情采集	0	99
编辑 复制 删除	2	商品CSV数据导入	csvPacketHelper	NULL	由淘宝助手或者甩手工具箱产生的CSV数据包直接导入到iWebShop系统中	0	99
编辑 复制 删除	4	QQ客服插件	sonline	{"color":"blue","DefaultsOpen":"true","Position":"...	网站和商家可以分别设置多个QQ客服，在商家主页，商品详情页面都有QQ咨询按钮	1	99
编辑 复制 删除	5	微信插件	wechat	{"wechat_Token":"11","wechat_AppID":"1111","wechat...	微信免登录免注册，微信支付，公众账号菜单定制，js-sdk对接	1	99
编辑 复制 删除	6	SCWS商品分词插件	words	[]	1,商品添加修改时对名称进行分词; 2,商品列表查询分词	1	99

全选 / 全不选 选中项: 修改 删除 导出

存进去的信息通过父类接口方法获取，后面会讲到。

3. 必须实现事件注册（挂接） public function reg(){事件注册代码...}

一个插件要想正常工作就必须注册（挂接）到系统的事件中，插件中的 reg 方法就是实现此插件的运行事件，是插件和系统的对接桥梁；同时这个 reg 方法也是插件的自动最先运行的初始化方法，只要是这个插件安装到系统并且为“开启”状态，那么当 iWebShop 运行的时候，就会自动调用这个方法实现插件的注册和初始化。所以说这个方法也是整个插件的入口和运行切入点，具有重要意义！

这里就要用到之前讲过的事件钩子了，我们先看看例子:

```
public function reg()
{
    //后台管理
    plugin::reg("onSystemMenuCreate",function(){
        $link = "/plugins/system_online_edit";
        Menu::$menu["插件"]["插件管理"][$link] = $this->name();
    });
    plugin::reg("onBeforeCreateAction@plugins@system_online_edit",function(){
        self::controller()->system_online_edit = function(){$this->online_edit();};
    });
    plugin::reg("onBeforeCreateAction@plugins@online_update",function(){
        self::controller()->online_update = function(){$this->online_update();};
    });

    //商家管理
    plugin::reg("onSellerMenuCreate",function(){
        $link = "/seller/seller_online_edit";
        menuSeller::$menu["配置模块"][$link] = $this->name();
    });
    plugin::reg("onBeforeCreateAction@seller@seller_online_edit",function(){
        self::controller()->seller_online_edit = function(){$this->online_edit();};
    });
    plugin::reg("onBeforeCreateAction@seller@online_update",function(){
        self::controller()->online_update = function(){$this->online_update();};
    });

    //前台显示部分
    plugin::reg("onServiceButton",$this,"showButton");
    plugin::reg("onFinishView@site@index",$this,"showFloat");
    plugin::reg("onFinishView@site@products",$this,"showFloat");
    plugin::reg("onFinishView@site@home",$this,"showFloat");
}
```

这个插件是 sonline（QQ 客服插件），这里面的 reg 函数注册了很多个事件，有些事件是之前讲过的系统预留事件，比如：onBeforeCreateAction@plugins@system_online_edit （当调用 plugins/system_online_edit 方法时候运行函数），在函数中用了 self::controller()接口可以获取当前运行

的控制器，给相应事件的控制器动态的增加动作（action），从而可以实现热插拔，比如之前 controller/plugins.php 控制器根本就没有 function system_online_edit，但是通过给控制器属性赋值“system_online_edit”self::controller()->system_online_edit = function(){.....};后 plugins 控制器就有这个方法了，可以在程序里面直接调用或者使用了。

我们通过这种匿名函数的方式，为 iWebShop 的控制器动态扩展方法。

onSystemMenuCreate 事件就是在 classes/menu.php 文件中，创建生成系统后台管理菜单时候触发的事件，这里注册事件里面的函数动态的把系统菜单进行了添加。

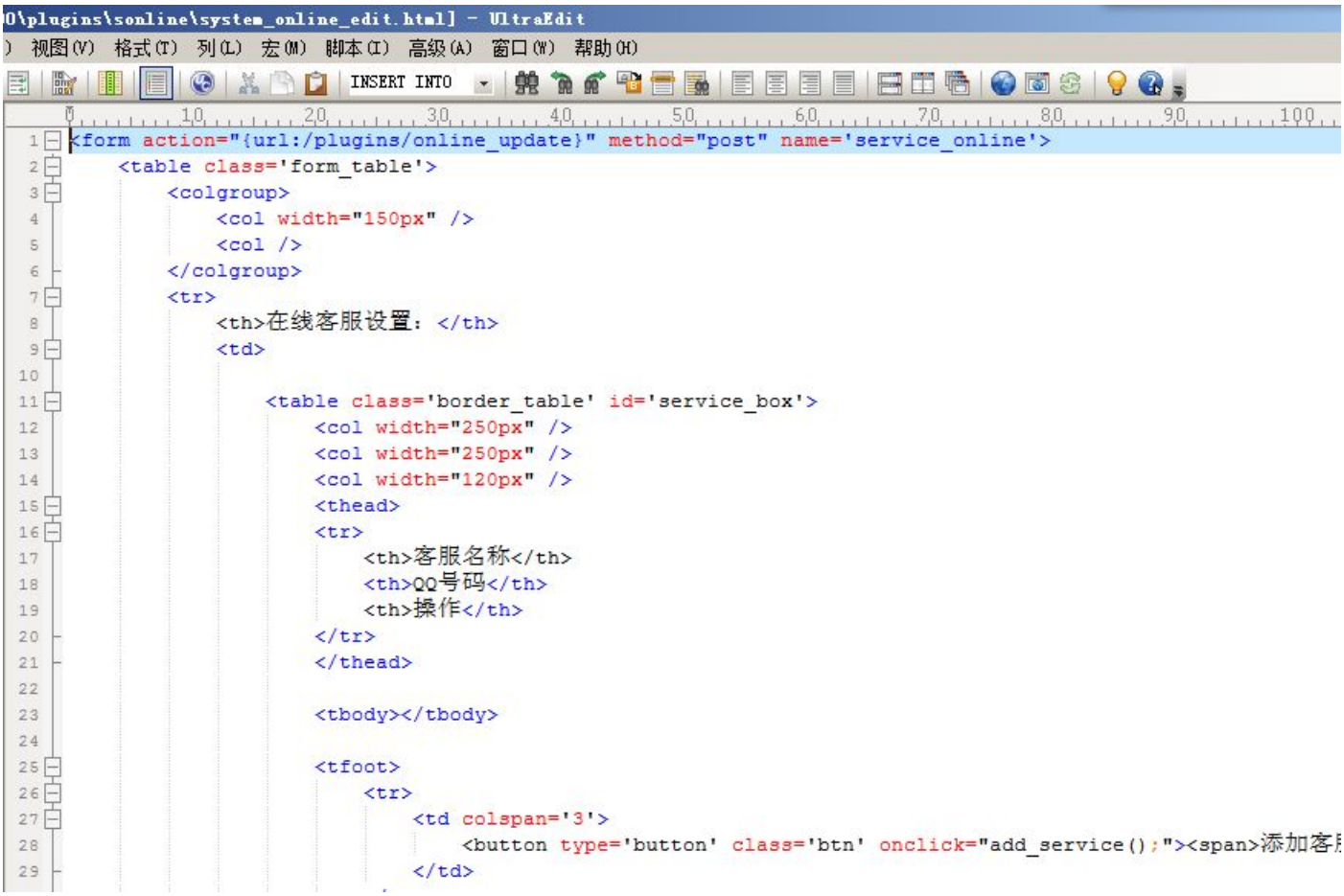
插件视图引入

插件开发的视图渲染默认情况下为引入模式（include 方式），即网址 URL 不会改变的，如果需要浏览器做非 include 的方式，需要在相应视图接口 redirect 里面增加第二个参数：true 即可实现。

很多情况下我们需要引入插件的视图来做 UI，引入视图分为 2 种类型，即插件基类中的 redirect 和 view 具体语法参考上文中的“插件基类公开接口”，需要注意的是，这 2 个接口如果要向模板里面传递参数都要追加第二个参数，数据类型 Array，当前控制器会自动调用$this->setRenderData($data);把$data 里面的 key 变成模板中的变量，关于这种数据的渲染方式可以参考《iWebShop 内核技术》

（1）redirect 接口：把插件视图融入当前的控制器 layout 中，比如后台管理，我们需要拥有公共部分的同时加入插件视图（模板）。比如 sonline（QQ 客服插件）的后台管理视图 \plugins\sonline\system_online_edit.html

另外 redirect 接口当其第二个参数为 true 的时候表示跳转 URL 地址路径而非 include 的形式。



模板里面就是一个 form 和 table 简单 html 代码片段，但是通过接口：

$this->redirect('system_online_edit');

就实现了融入当前控制器视图的功能，看起来插件和整个控制器都是一体的。

（2）view 接口：可以做独立的模板代码片段嵌入，不包括当前控制器的 layout。比如：
$this->view('system_online_edit');

以上 2 种视图渲染方式可以根据需要自己做切换。